첫페이지

- **프로젝트명:**000
- **팀명 / 팀원 / 담당 역할**
- **프로젝트 요약:**"OpenAI + Django 기반 자격증 데이터 통계 및 AI 추천 API 개발"
- **시스템 구성도:**전체 구조가 한눈에 보이는 간단한 아키텍처 다이어그램

발표 시작 시, 프로젝트의 전체 구성을 한눈에 보여주는 시각 자료로 활용하세요.

기술 완성도

2-1. **API 기능 구현**

- 주요 기능 목록 (CRUD, 인증, 연동 흐름)
- 주요 엔드포인트 예시 (/api/certificates/, /api/stats/)
- 예외 처리 및 인증 흐름 (JWT, DRF Permissions)
- Swagger 캡처 이미지 또는 crul 테스트

작성예시:

① 주요 기능 목록 (CRUD, 인증, 연동 흐름)

Django REST Framework를 기반으로 CRUD, JWT 인증, AI 연동 기능을 구현했습니다.
아래 표의 4개 핵심 기능은 Swagger, Insomnia 또는 curl로 실제 동작을 검증했습니다.

기능 구분	설명	대표 엔드포인트	테스트 여부
회원가입/로그인	JWT 기반 인증 구현	/api/auth/register, /api/auth/login	✓
자격증 CRUD	자격증 등록·조회·수정·삭제	/api/certificates/	✓
통계 조회	응시자수, 합격률 통계	/api/stats/	✓
AI 추천	사용자 입력 기반 자격증 추천	/api/ai/recommend/	✓

캡처: Swagger(또는 Insomnia)에서 위 4개 엔드포인트가 보이는 전체 목록 1장

② 주요 엔드포인트 예시 (정상 동작 증명)

핵심 API 3건의 정상 동작을 200 OK + JSON 응답으로 증명합니다.

```
# 예시
GET /api/certificates/

응답:
[
  {"id":1,"name":"정보처리기사","category":"IT"},
  {"id":2,"name":"빅데이터분석기사","category":"Data"}
]
```

캡처: 200 OK 응답 화면 캡처 1장
(또는 curl -X GET http://localhost:8000/api/certificates/실행 결과 화면)
목록조회(GET), 수정(PUT), 조회(GET)등도 위와 같은 방법으로 Json또는 curl로 증명해서 성공캡처를 각각 만듦

③ 예외 처리 및 인증 흐름
JWT 인증이 필요한 API에서는 토큰이 없거나 유효하지 않을 경우 일관된 JSON 에러를 반환합니다.

구분	요청	응답 예시
정상 로그인	/api/auth/login	{ "access":"<JWT_ACCESS>", "refresh":"<JWT_REFRESH>" }
인증 실패	보호 API 호출 (토큰 없음)	{ "detail": "인증 자격 증명이 제공되지 않았습니다." }

2-2. **AI 모델/연동 품질**

- 모델 개요 (모델명, 데이터셋, 목적)
- OpenAI 또는 직접 학습 모델 연동 흐름도
- 결과 예시 스크린샷 (입력/출력)
- 모델 성능 간단 비교표

작성예시:

① 모델 개요
사용자의 관심 키워드를 입력받아 관련 자격증을 추천합니다.
OpenAI GPT-4o-mini 모델을 사용했으며, 도메인 예시와 룰을 프롬프트에 포함해 정확도를 보완했습니다.

항목	내용 (예시)
모델명	OpenAI GPT-4o mini (또는 사내/로컬 모델명)
목적	사용자 관심 키워드 → 자격증 추천/설명 생성
입력	{ "interest": "데이터 분석" }
출력	["빅데이터분석기사","ADsP","SQLD"]+ 간략 설명

② 연동 흐름도
graph LR
A[Client / Frontend] --> B[DRF Endpoint: POST /api/ai/recommend]
B --> C[Prompt Builder (관심사→프롬프트)]
C --> D[LLM API 호출 (timeout, retries)]
D --> E[응답 파싱/정렬 (Top-N)]

E --> F[응답 캐시 (선택)]
F --> G[JSON 응답 반환]

③ 결과 예시 (입력/출력)
입력과 출력의 실제 예시를 200 OK JSON과 함께 캡처해 모델 동작을 증명합니다.

```
# 예시 1: 요청/응답(JSON)
요청
POST /api/ai/recommend
{"interest":"데이터 분석"}

응답(200)
{
  "recommendations":[
    {"name":"빅데이터분석기사","reason":"데이터 처리·분석 전반"},
    {"name":"ADsP","reason":"기초 데이터 이해·분석"},
    {"name":"SQLD","reason":"데이터 질의·모델링"}
  ]
}

# 예시 2: curl 대체 가능
curl -X POST http://localhost:8000/api/ai/recommend/ ₩
  -H "Content-Type: application/json" ₩
  -d '{"interest":"보안"}'
```

캡처: Swagger/Insomnia의 요청/응답 화면, 터미널 curl 200 OK 화면

2-3. **인프라 구성 (Docker/K8s)**

- Dockerfile, docker-compose 요약 코드
- Kubernetes 구성도 (Deployment / Service / Ingress / HPA)
- 배포 구조 다이어그램 (개발 → 빌드 → 배포)

작성예시:

① Dockerfile & docker-compose 요약 코드

Docker를 이용해 Django 앱과 MySQL을 컨테이너로 구성했습니다.

Gunicorn 서버로 서비스하며, 로컬과 EC2에서 동일하게 실행 가능합니다.

```
docker-compose up
web_1 | Application startup complete.
```

캡처: Docker 실행 로그 화면 캡처

② Kubernetes 구성도 (Deployment / Service / Ingress / HPA)

Docker 이미지를 Kubernetes로 배포했습니다.

Deployment → Service → Ingress 순서로 구성하고, HPA로 CPU 부하 시 자동 확장되도록 설정했습니다.

```
NAME                READY     STATUS
django-app-1     1/1         Running
django-svc         ClusterIP  10.96.12.34
```

캡처: kubectl get pods,svc실행 결과 캡처

③ 배포 구조

GitHub Actions를 통한 CI/CD 자동화

개발자가 커밋 → Docker 빌드 → 이미지 푸시 → 서버 배포

캡처: Actions의 ✓ build & deploy success 로그 또는 docker ps결과 캡처

성능 및 모니터링

3-1. **부하 테스트 결과**

- k6 테스트 코드 스니펫
- Grafana 지표
- 개선 전후 성능 비교표

작성예시:

① k6 테스트 코드

k6로 /api/certificates//api/stats/엔드포인트에 대해 동시 요청 부하 테스트를 수행했습니다. 평균 응답시간과 오류율을 측정했습니다.

```
import http from 'k6/http';
export default function () {
  let res = http.get('http://localhost:8000/api/certificates/');
}
```

캡처: ✓ status is 200표시된 k6 실행 결과 캡처

② Grafana 지표

Prometheus + Grafana를 연동해 API 응답 지연(p95/p99), RPS(Request per Second), 오류율(5xx)을 모니터링했습니다.

예시 수치 (요약)

- RPS: 평균 110 req/s
- p95 응답시간: 280 ms
- p99 응답시간: 420 ms
- 5xx 에러율: 0.5% 이하

캡처: Grafana 대시보드에서 "RPS / Latency / Error Rate" 그래프가 보이는 캡처

③ 개선 전후 성능 비교표

Gunicorn 워커 수 조정 및 N+1 쿼리 최적화 후 성능이 개선되었습니다

구분	RPS(요청/초)	p95(ms)	p99(ms)	에러율(%)
개선 전	45	480	760	2.1
개선 후	110	280	420	0.5

3-2. **모니터링 체계**

- Prometheus + Grafana 연동 구조도
- 주요 지표 (CPU, Memory, Latency)
- 알람 설정 예시 (Alertmanager 캡처)

작성예시:

① Prometheus + Grafana 연동

애플리케이션 컨테이너에 Prometheus Exporter (/metrics)를 설정하여 CPU, Memory, 응답 지연시간(Latency) 지표를 수집했습니다.
Prometheus 서버가 1분 주기로 데이터를 수집하고, Grafana 대시보드에서 실시간으로 시각화했습니다.

캡처:

- Prometheus Targets 페이지 (localhost:9090/targets)에서
  django_app /metrics → UP 상태가 보이는 화면 1장
- Grafana 대시보드 CPU · 메모리 그래프 1장

② 주요 지표 (CPU / Memory / Latency)

서비스 운영 중 주요 지표를 수집했습니다. 모든 지표는 Grafana 대시보드에서 실시간 모니터링 됩니다.

구분	지표 명	정상 범위	비고
서버 부하	CPU Usage	< 70 %	HPA 확장 기준
메모리	Memory Usage (MB)	< 500 MB	컨테이너 기준
응답 속도	API Latency (p95)	< 300 ms	/api/certificates/
오류 비율	5xx Error Rate	< 1 %	에러 알람 트리거 조건

캡처: Grafana CPU/Memory/Latency 그래프

③ 알람 설정

Alertmanager로 CPU 70% 초과 시 Slack 알림 전송

캡처: Alertmanager 웹 또는 Slack 알림 화면 캡처

기획 및 분석

4-1. 주제 적합성
프로젝트 목적:
자격증 정보가 분산되어 있어 사용자가 자신에게 맞는 자격증을 찾기 어렵다는 문제를 해결

왜 이 프로젝트를 만들었는가?
이 주제가 어떤 사회적·산업적 문제를 해결하려는 것인가?
현실에서 어디에 쓸 수 있는가?
즉, 기술적 구현이 아니라 기획 의도와 실무 가치를 설명
아이디어의 타당성을 평가하는 부분입니다.

4-2. **요구사항 분석**

- ERD 다이어그램
- 논리적 흐름(Usecase Diagram)

작성예시:

① **ERD 다이어그램:**User, Certificate, Stats, RecommendLog 관계 시각화
프로젝트의 주요 모델은 사용자, 자격증, 통계, 추천 로그로 구성되며, 각 모델 간 관계를 ERD로 표현했습니다.

② 논리적 흐름 (Usecase Diagram) 표로 구현

기능	Method	Endpoint	설명
자격증 조회	GET	/api/certificates/	자격증 목록 반환
통계 조회	GET	/api/stats/	합격률·응시자수 반환
AI 추천	POST	/api/ai/recommend/	관심분야 기반 자격증 추천

4-3. **AI 적용 계획**

- AI 적용 목적
- 결과 해석 및 향후 개선 방향

작성예시:

① AI 적용 목적

사용자의 관심 분야(예: 데이터, 디자인, 보안 등)를 입력받아

해당 분야에 적합한 자격증을 추천하기 위해 AI 모델(OpenAI GPT API)을 적용했습니다.

단순 키워드 매칭이 아닌 문맥 기반 추천을 통해 사용자의 진로 방향에 맞는 자격증을 제안하는 것을 목표로 합니다.

항목	내용
모델명	OpenAI GPT-4o-mini
적용 위치	/api/ai/recommend/API
입력 데이터	사용자가 입력한 관심 키워드 (예: "데이터 분석")
출력 결과	추천 자격증 목록 (예: "빅데이터분석기사", "SQLD", "ADsP")
적용 목적	사용자의 목표/분야에 맞는 자격증을 자동 추천

캡처:

Swagger나 Insomnia에서 /api/ai/recommend/요청 → JSON 추천 결과(200 OK) 캡처 1장

(입력 "interest":"보안"→ 출력 ["정보보안기사","CISSP","CEH"]식으로)

협업 및 코드 품질

5-1. **협업 체계**

- GitFlow 전략:main → develop → feature
- 커밋 메시지 규칙 예시 feat:, fix:, refactor:형식
- 역할 분담표

작성예시:

① Git 브랜치 전략 (GitFlow)

GitFlow 전략을 기반으로 브랜치를 관리했습니다.
main브랜치는 배포용, develop은 통합 개발용으로 사용하고,
기능 단위로 feature/브랜치를 분리하여 협업했습니다.

브랜치 구조 예시

```
main
  └── develop
        ├── feature/auth
        ├── feature/certificate
        ├── feature/ai-recommend
        └── hotfix/stats-bug
```

캡처:

GitHub 브랜치 목록 화면 캡처
(예: main, develop, feature/...들이 보이게)
또는 Pull Request 화면 캡처 1장
(예: "Merge feature/ai-recommend → develop")
기능 단위 브랜치 생성 → 개발 완료 후 PR 리뷰 → develop 병합 → main 배포 순으로 관리했습니다.

역할분담표

이름	역할	담당
000	팀장	API, DB 설계
000	AI	모델 연동
000	인프라	Docker, Grafana
000	문서	Swagger, 발표자료

5-2. **코드 구조/가독성**

- 폴더 구조 트리 (tree -L 2)
- 일관된 네이밍 규칙 표

작성예시:

① 폴더 구조 트리 (tree –L 2)

기능별로 앱(app) 단위로 코드를 구성하고,

Django의 표준 MVC 구조(Model–View–Template)를 유지했습니다.

api/, core/, users/, certificates/등 각 앱은 독립적으로 관리되어 유지보수가 용이합니다.

폴더 디렉토리

```
myproject/
├── manage.py
├── config/
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── users/
│   ├── models.py
│   ├── serializers.py
│   ├── views.py
│   └── urls.py
├── certificates/
│   ├── models.py
│   ├── views.py
│   ├── serializers.py
│   └── urls.py
├── ai/
│   ├── views.py
│   └── utils.py
└── requirements.txt
```

VSCode나 터미널에서 tree -L 2실행 결과 캡처
(또는 VSCode 왼쪽 탐색기 창 전체 화면 캡처)

② 일관된 네이밍 규칙 표
클래스, 함수, 파일명, 변수명을 통일된 규칙으로 관리하여
코드 가독성과 유지보수성을 높였습니다.

항목	규칙	예시
클래스명	PascalCase	CertificateViewSet, UserSerializer
함수명	snake_case	get_certificates(), create_user()
파일명	소문자 + 언더바	views.py, serializers.py
URL 경로	복수형 명사 사용	/api/certificates/, /api/users/
변수명	명확한 의미 중심	user_count, pass_rate

캡처:
VSCode에서 코드 일부 캡처 (예: views.py 내 함수 이름들) 클래스명·함수명·URL이 위 표처럼 일관된 형식으로 보이면 충분

5-3. **테스트 및 에러 핸들링**

- pytest 실행 통과 캡처
- 테스트 커버리지
- 예외처리 예시 코드

작성예시:

① pytest 실행 캡처

주요 API(View)와 모델 로직에 대해 pytest 단위 테스트를 작성하여 기능을 검증했습니다.
테스트 실행 시 모든 케이스가 통과됨을 확인했습니다.

테스트 실행 예시 코드 (요약)

```
# tests/test_certificate_api.py
import pytest
from rest_framework.test import APIClient

@pytest.mark.django_db
def test_get_certificates_list():
    client = APIClient()
    response = client.get("/api/certificates/")
    assert response.status_code == 200
```

캡처:

```
==================== test session starts ====================
collected 5 items
tests/test_certificate_api.py .....                    [100%]
==================== 5 passed in 1.22s ======================
```

주요 API 테스트 5건이 모두 성공적으로 통과되었습니다.

② 테스트 커버리지 표

pytest-cov 플러그인을 사용해 코드 커버리지를 측정했습니다.

전체 코드의 85% 이상이 테스트로 검증되었습니다

실행명령어 예시

pytest --cov=certificates —cov=users --cov-report=term-missing

테스트 커버리지 결과 예시

모듈	라인수	커버된 라인	커버리지(%)
certificates/views.py	120	108	90%
certificates/models.py	80	74	92%
users/views.py	100	83	83%
전체 평균	—	—	**88%**

캡처:

터미널에 표시된 커버리지 요약표 화면

TOTAL 300 266 88%

③ 예외처리 예시 코드

API 실행 중 발생할 수 있는 예외 상황(데이터 없음, 인증 오류, 외부 API 실패 등)에 대해 일관된 에러 응답 형식을 제공하도록 처리했습니다

예시코드

```
# certificates/views.py
from rest_framework.views import exception_handler

def custom_exception_handler(exc, context):
    response = exception_handler(exc, context)
    if response is not None:
        response.data['status_code'] = response.status_code
        response.data['message'] = response.data.get('detail', '오류가 발생했습니다.')
    return response
```

예외 응답 예시

```
{
  "status_code": 404,
  "message": "요청하신 자격증을 찾을 수 없습니다."
}
```

캡처:

실제 404 오류 발생 시 Swagger/Insomnia 응답 화면
(JSON과 같은 구조가 보이게)
또는 custom_exception_handler()코드 일부가 보이는 VSCode 화면

README 문서

6-1. **기술 문서/README**

- README 주요 섹션 요약 (설치 / 실행 / ERD / API 명세)
- 아키텍처/ERD 이미지 삽입

이 파트는 README를 작성한후 작성한 화면을 캡처하여 증빙으로 삽입하면 됩니다.

README 주요 섹션 구성 예시

섹션	설명
프로젝트 개요	프로젝트 목적, 주요 기능 요약
설치 방법	git clonepip install -r requirements.txt.env설정
실행 방법	python manage.py runserveror docker-compose up
ERD	DB 구조 이미지(erd.png) 첨부
API 명세	주요 엔드포인트 정리(GET /api/certificates/, POST /api/ai/recommend/)
AI 적용 내용	OpenAI 연동 방법 요약
팀 정보 및 역할	팀 구성 및 역할 분담 정리
결과 및 시연 영상	실행 화면 캡처 또는 영상 링크

참고: 리드미에 아키텍처도 삽입해 주세요.

6-2. **팀워크/커뮤니케이션**

- 협업 기록 (회의/노션/피그마 일부 캡처)